Trabalho prático 2
Sistemas Operativos

Bruno Bastos 93302

Eduardo Santos 93107

Conteúdo

Introdução 3
Estruturas de dados 4
Entidade Agent 6
Entidade Watcher 9
Entidade Smoker 14
Resultados obtidos 19
Conclusão 20
Bibliografia 20

Introdução

O segundo trabalho prático de Sistemas Operativos consiste numa aplicação em C que, através da utilização de semáforos, simula o processo necessário para que um fumador fume um cigarro.

Este problema tem como objetivos compreender a utilização de semáforos assim como os mecanismos associados à execução e sincronização de processos e threads.

Será necessário alterar o código base fornecido pelo docente em 3 ficheiros correspondentes às entidades: smoker, watcher e agent.

Estruturas de dados

Neste problema existem três estruturas de dados bastante importantes: FULL_STAT, STAT e SHARED_DATA. A estrutura STAT está incorporada na estrutura FULL_STAT e contém as informações dos estados de cada entidade do problema. A estrutura FULL_STAT além da estrutura STAT tem também as variáveis partilhadas usadas neste problema. Por fim a estrutura SHARED_DATA contém a estrutura FULL_STAT e também os semáforos usados na resolução deste problema.

```
typedef struct {
    /** \brief agent state */
    unsigned int agentStat;
    /** \brief watchers state */
    unsigned int watcherStat[NUMINGREDIENTS];
    /** \brief smokers state */
    unsigned int smokerStat[NUMSMOKERS];

} STAT;

typedef struct
{   /** \brief state of all intervening entities */
    STAT st;

    /** \brief number of ingredients */
    int nIngredients;

    /** \brief number of orders to be performed by agent (each order incl
udes a pack of 2 ingredients) */
    int nOrders;

    /** \brief number of smokers */
    int nSmokers;

    /** \brief flag used by agent to close factory */
    bool closing;

    /** \brief inventory of ingredients */
    int ingredients[NUMINGREDIENTS];

    /** \brief number of ingredients already reserved by watcher */
    int reserved[NUMINGREDIENTS];

    /** \brief number of cigarettes each smoker smoked */
    int nCigarettes[NUMSMOKERS];

} FULL_STAT;
```

```
typedef struct
{ /** \brief full state of the problem */
    FULL_STAT fSt;

    /* semaphores ids */
    /** \brief identification of critical region protection semaphore – v
al = 1 */
    unsigned int mutex;
    /** \brief identification of semaphore used by watchers to wait for a
gent - val = 0 */
    unsigned int ingredient[NUMINGREDIENTS];
    /** \brief identification of semaphore used by agent to wait for smok
er to finish rolling - val = 0 */
    unsigned int waitCigarette;
    /** \brief identification of semaphore used by smoker to wait for wat
chers – val = 0  */
    unsigned int wait2Ings[NUMSMOKERS];

} SHARED_DATA;
```

Foi feita uma tabela com informação sobre os semáforos, quais entidades fazem Up ou Down e em quais funções isso acontece.

	Entidade		Funções	
	Up	Down	Up	Down
mutex	Todas	Todas	Todas	Todas
wait2Ings[id]	Watcher	Smoker	waitForIngredient/ informSmoker	waitForIngredients
waitCigarette	Smoker	Agent	rollingCigarette	waitForCigarette
Ingredients[id]	Agent	Watcher	prepareIngredients/ closeFactory	waitForIngredient

Entidade Agent

O agent produz os ingredientes necessários para que o smoker consiga fumar. O agent só volta a produzir mais ingredientes após o smoker ter fumado o seu cigarro. Esta entidade tem 3 funções: prepareIngredients, waitForCigarrete e closeFactory. No seu ciclo de vida o Agent produz ingredientes suficientes para 5 cigarros e depois fecha a fábrica.

```
/* simulation of the life cycle of the agent */

    int nOrders=0;
    while(nOrders < sh->fSt.nOrders) {
       prepareIngredients();
       waitForCigarette();

       nOrders++;
    }

    closeFactory();
```

prepareIngredients()

Nesta função o Agent produz dois ingredientes aleatórios e entra na zona critica (Down no mutex) para alterar o seu estado para PREPARING (o estado é guardado através da função saveState()) e o inventário de ingredientes, pois este é uma variável partilhada. Depois sai da zona critica (Up no mutex) e da semUp nos semáforos ingredients[id] onde o id corresponde ao id dos ingredientes produzidos.

```
static void prepareIngredients ()
{
    int random1 = rand()%3;
    int random2;
    while((random2=rand()%3)==random1){}
    if (semDown(semgid, sh->mutex)==1) {
/* enter critical region */
        perror ("error on the up operation for semaphore access (AG)");
        exit (EXIT_FAILURE);
    }
    /* TODO: insert your code here */
    sh->fSt.st.agentStat=PREPARING;
    saveState(nFic,&sh->fSt);
    sh->fSt.ingredients[random1]++;
```

```
    sh->fSt.ingredients[random2]++;
    if (semUp (semgid, sh->mutex)==- 1) {
 /* leave critical region */
        perror ("error on the up operation for semaphore access (AG)");
        exit (EXIT_FAILURE);
    }
    /* TODO: insert your code here */
    if(semUp(semgid,sh->ingredient[random1])==-1){
        perror ("error on the up operation for semaphore access (AG)");
        exit (EXIT_FAILURE);
    }
    if(semUp(semgid,sh->ingredient[random2])==-1){
        perror ("error on the up operation for semaphore access (AG)");
        exit (EXIT_FAILURE);
    }
}
```

waitForCigarette()

Nesta função é feito um Down no mutex dando assim entrada na zona critica para poder alterar o estado do Agent para WAITING_CIG. Esse estado é guardado através da função saveState(). Esta função só acaba quando o Smoker acabar de enrolar o cigarro. Isso é feito através de um Down no semáforo waitCigarette.

```
static void waitForCigarette ()
{
    if (semDown (semgid, sh->mutex) == -
1) {                                                    /* enter critical region */
        perror ("error on the up operation for semaphore access (AG)");
        exit (EXIT_FAILURE);
    }
    /* TODO: insert your code here */
    sh->fSt.st.agentStat=WAITING_CIG;
    saveState(nFic,&sh->fSt);

    if(semUp(semgid,sh->mutex)==-
1){                                                 /* leave critical region */
        perror ("error on the up operation for semaphore access (AG)");
        exit (EXIT_FAILURE);
    }
    /* TODO: insert your code here */
    if(semDown(semgid,sh->waitCigarette)==-1){
        perror ("error on the up operation for semaphore access (AG)");
        exit (EXIT_FAILURE);
    }
}
```

closeFactory()

Após entrar na região critica através de um Down no mutex, o Agent muda e o seu estado para CLOSING_A e altera o valor da variável partilhada closing para true. Depois de guardar através da função saveState() o Agent sai da região partilhada fazendo um Up no mutex. Já fora da região critica o Agent dá up a todos os semáforos do array de semáforos ingredient para notificar os watchers que a fábrica vai fechar.

```
static void closeFactory ()
{
    if (semDown (semgid, sh->mutex) == -
1) {                                                    /* enter critic
al region */
        perror ("error on the up operation for semaphore access (AG)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.agentStat=CLOSING_A;
    sh->fSt.closing=true;
    saveState(nFic,&sh->fSt);

    if (semUp (semgid, sh->mutex) == -
1) {                                                      /* leave crit
ical region */
        perror ("error on the up operation for semaphore access (AG)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    for(int i=0;i<NUMSMOKERS;i++){
        if(semUp(semgid,sh->ingredient[i])==-1){
            perror ("error on the up operation for semaphore access (AG)"
);
            exit (EXIT_FAILURE);
        }
    }
}
```

Entidade Watcher

O watcher é a entidade responsável pela atribuição dos ingredientes produzidos pelo agent aos smokers. Cada watcher é responsável por um smoker e é o watcher que indica ao smoker se ele pode fumar assim como se fábrica vai fechar. Após o agent ter produzido os ingredientes cabe ao watcher saber qual dos smokers tem ingredientes suficientes para fumar. O ciclo de vida do watcher acaba quando a fábrica fechar.

```
/* simulation of the life cycle of the watcher */
    int id = n, smokerReady;
    while( waitForIngredient (id) ) {
        smokerReady = updateReservations(id);
        if(smokerReady>=0) informSmoker(id, smokerReady);
    }
```

O watcher tem 3 funções: waitForIngredient, updateReservations e informSmoker.

informSmoker()

Nesta função, o watcher entra na região critica através de um Down no mutex e altear o seu estado para INFORMING e sai da região após o guardar. Depois notifica o smoker através de um Up no semáforo wait2Ings[id] em que o id corresponde ao smoker que pode fumar.

```
static void informSmoker (int id, int smokerReady)
{
    if (semDown (semgid, sh->mutex) == -
1)  {                                                    /* enter critic
al region */
        perror ("error on the down operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }
    sh->fSt.st.watcherStat[id] = INFORMING;
    saveState(nFic, &sh->fSt);
    if (semUp (semgid, sh->mutex) == -
1) {                                                      /* exit crit
ical region */
        perror ("error on the up operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }
    if (semUp (semgid, sh->wait2Ings[smokerReady]) == -1)  {
        perror ("error on the up operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }
}
```

waitForIngredient()

Esta função retorna um boolean (ret) que retorna false caso a fábrica esteja a encerrar. O watcher entra na região critica e altera o seu estado para WAITING_ING e guarda esse valor. Sai da região critica e espera que o agent prepare os ingredientes através de um Down no semáforo ingredient[id]. Entra novamente na região critica e verifica se a fábrica esta a fechar. Se isto se verificar o watcher muda a variável ret para false e o seu estado para CLOSING_W e de seguida sai da região critica para notificar o smoker de que a fábrica esta a fechar através de um Up no semáforo wait2Ings. Retorna o valor da variável ret.

```
static bool waitForIngredient(int id)
{
    bool ret=true;

    if (semDown (semgid, sh->mutex) == -
1)  {                                                   /* enter critic
al region */
        perror ("error on the down operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */

    // updates state
    sh->fSt.st.watcherStat[id] = WAITING_ING;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -
1) {                                                      /* exit crit
ical region */
        perror ("error on the up operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */


    // waits agent
    if (semDown (semgid, sh->ingredient[id]) == -1)  {
        perror ("error on the down operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }
```

```
    if (semDown (semgid, sh->mutex) == -
1)  {                                                    /* enter critic
al region */
        perror ("error on the down operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */

    if (sh->fSt.closing) {
        // updates state
        sh->fSt.st.watcherStat[id] = CLOSING_W;
        saveState(nFic, &sh->fSt);

        ret = false;
    }

    if (semUp (semgid, sh->mutex) == -
1) {                                                      /* exit crit
ical region */
        perror ("error on the up operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

    if (sh->fSt.closing) {

        // notifies smoker
        if (semUp (semgid, sh->wait2Ings[id]) == -1)  {
            perror ("error on the up operation for semaphore access (WT)"
);
            exit (EXIT_FAILURE);
        }
    }

    return ret;

}
```

updateReservations()

Nesta função a variável de retorno deverá ser igual ao id do smoker que tem os ingredientes todos para começar a enrolar. Caso nenhum dos smokers esteja apto para enrolar a função retorna -1. O watcher começa por entrar na região critica e altera o seu estado para UPDATING. Depois verifica quais dos ingredientes foram produzidos pelo agent de acordo com o semáforo ingredient[id]. O watcher guarda na variável partilhada reserved[id] caso o ingrediente tenha sido produzido pelo agent. Depois verifica qual dos smokers tem os ingredientes suficientes para estar apto a fumar e atribui à variável de retorno o id desse smoker. Após isso, sai da região critica e retorna a variável ret.

```
static int updateReservations (int id)
{
    int ret = -1;

    if (semDown (semgid, sh->mutex) == -
1)  {                                                   /* enter critic
al region */
        perror ("error on the down operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */

    sh->fSt.st.watcherStat[id] = UPDATING;
    saveState(nFic, &sh->fSt);

    if (sh->fSt.ingredients[id] > 0)
        sh->fSt.reserved[id]++;

    // checks if some smoker may start rolling a cigarette
    if (sh->fSt.reserved[HAVETOBACCO] && sh->fSt.reserved[HAVEMATCHES]) {
        sh->fSt.reserved[HAVETOBACCO]--;
        sh->fSt.reserved[HAVEMATCHES]--;
        ret = PAPER;
    }
    else if (sh->fSt.reserved[HAVETOBACCO] && sh->fSt.reserved[HAVEPAPER]
) {
        sh->fSt.reserved[HAVETOBACCO]--;
        sh->fSt.reserved[HAVEPAPER]--;
        ret = MATCHES;
    }
    else if (sh->fSt.reserved[HAVEMATCHES] && sh->fSt.reserved[HAVEPAPER]
) {
```

```
        sh->fSt.reserved[HAVEMATCHES]--;
        sh->fSt.reserved[HAVEPAPER]--;
        ret = TOBACCO;
    }

    if (semUp (semgid, sh->mutex) == -
1) {                                                    /* exit crit
ical region */
        perror ("error on the up operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

    return ret;
}
```

Entidade Smoker

Existem 3 smokers e cada tem uma fonte infinita de um dos 3 ingredientes necessários para poder fumar um cigarro. Cada smoker tem um watcher responsável por ele. Após o agent produzir 2 ingredientes aleatórios, um dos smokers está apto para fumar. O watcher responsável avisa o smoker que ele pode começar a enrolar. Depois de acabar de enrolar o smoker avisa o agent para voltar a produzir mais ingredientes. O ciclo de vida do smoker consiste na espera dos ingredientes, enrolar o cigarro e fumá-lo.

```
/* simulation of the life cycle of the smoker */
    while(waitForIngredients(n)) {
        rollingCigarette(n);
        smoke(n);
    }
```

Para isso o smoker tem 3 funções: waitForIngredients, rollingCigarette e smoke.

waitForIngredients()

Esta função vai retornar verdadeiro se os ingredientes estiverem disponíveis e falso caso a fabrica esteja para fechar. O smoker começa por entrar na região critica, muda o seu estado para WAITING_2ING e sai depois de o guardar. Posteriormente, fica à espera que o watcher o notifique através do semáforo wait2Ings[id]. Volta a entrar na região critica caso seja notificado e verifica se a fabrica vai fechar através da variável partilhada closing. Caso isto se verifique muda o seu estado para CLOSING_S, guarda-o e muda a variável de retorno para false. Se isto não se verificar, o smoker vai remover do inventário ingredientes, os ingredientes que usou precisa para fumar. Sai da região critica e retorna a variável ret.

```
static bool waitForIngredients (int id)
{
    bool ret = true;

    if (semDown (semgid, sh->mutex) == -
1)  {                                                   /* enter critic
al region */
        perror ("error on the down operation for semaphore access (SM)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */

    sh->fSt.st.smokerStat[id] = WAITING_2ING;
    saveState(nFic, &sh->fSt);


    if (semUp (semgid, sh->mutex) == -
1) {                                                      /* exit crit
ical region */
        perror ("error on the up operation for semaphore access (SM)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */

    if (semDown (semgid, sh->wait2Ings[id]) == -1)  {
        perror ("error on the down operation for semaphore access (SM)");
        exit (EXIT_FAILURE);
    }

    if (semDown (semgid, sh->mutex) == -
1)  {                                                   /* enter critic
al region */
        perror ("error on the down operation for semaphore access (SM)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */

    if (sh->fSt.closing) {
        // updates state
        sh->fSt.st.smokerStat[id] = CLOSING_S;
        saveState(nFic, &sh->fSt);

        ret = false;
    }
    else {
        // updates inventory
```

```
        switch (id) {
            case HAVETOBACCO:
                sh->fSt.ingredients[MATCHES]--;
                sh->fSt.ingredients[PAPER]--;
                break;
            case HAVEMATCHES:
                sh->fSt.ingredients[TOBACCO]--;
                sh->fSt.ingredients[PAPER]--;
                break;
            case HAVEPAPER:
                sh->fSt.ingredients[TOBACCO]--;
                sh->fSt.ingredients[MATCHES]--;
        }
    }

    if (semUp (semgid, sh->mutex) == -
1) {                                                    /* exit crit
ical region */
        perror ("error on the up operation for semaphore access (SM)");
        exit (EXIT_FAILURE);
    }

    return ret;
}
```

rollingCigarette()

É escolhido ao acaso o tempo que smoker vai demorar a enrolar o cigarro. Depois o smoker entra na região critica e altera o seu estado para ROLLING, guarda e sai. Demora o seu tempo a enrolar o cigarro e de seguida notifica o agent através de um Up no semáforo waitCigarette para que este comece a produzir mais ingredientes.

```
static void rollingCigarette (int id)
{
    double rollingTime = 100.0 + normalRand(30.0);

    if (semDown (semgid, sh->mutex) == -
1)  {                                                  /* enter critic
al region */
        perror ("error on the down operation for semaphore access (SM)");
        exit (EXIT_FAILURE);
    }
```

```
    /* TODO: insert your code here */

    sh->fSt.st.smokerStat[id] = ROLLING;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -
1) {                                                    /* exit crit
ical region */
        perror ("error on the up operation for semaphore access (SM)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */

    if (rollingTime > 0)
        usleep(rollingTime);

    if (semUp (semgid, sh->waitCigarette) == -1) {
        perror ("error on the up operation for semaphore access (SM)");
        exit (EXIT_FAILURE);
    }
}
```

smoke()

O tempo que o smoker demora a fumar é calculado ao acaso. O smoker começa por entrar na região critica, altera o seu estado para SMOKING e aumenta o numero de cigarros que fumou na variável partilhada nCigarettes[id]. Sai da região partilhada e leva o seu tempo a fumar o cigarro.

```
static void smoke(int id)
{
    double smokingTime = 100.0 + normalRand(30.0);

    if (semDown (semgid, sh->mutex) == -
1)  {                                                  /* enter critic
al region */
        perror ("error on the down operation for semaphore access (SM)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */

    sh->fSt.st.smokerStat[id] = SMOKING;
```

```
    saveState(nFic, &sh->fSt);

    // updates number of cigarretes smoked
    sh->fSt.nCigarettes[id]++;

    if (semUp (semgid, sh->mutex) == -
1) {                                                    /* exit crit
ical region */
        perror ("error on the up operation for semaphore access (SM)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */

    if (smokingTime > 0)
        usleep(smokingTime);
}
```

Resultados obtidos

Smokers - Description of the internal

AG	W00	W01	W02	S00	S01	S02	I00	I01	I02	C00	C01	C02
1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	1	0	0	0
2	0	0	0	0	0	0	0	1	1	0	0	0
2	0	0	0	0	0	0	0	1	1	0	0	0
2	0	0	0	0	0	0	0	1	1	0	0	0
2	0	0	0	0	0	0	0	1	1	0	0	0
2	0	0	0	0	0	0	0	1	1	0	0	0
2	0	0	0	0	0	0	0	1	1	0	0	0
2	0	0	1	0	0	0	0	1	1	0	0	0
2	0	1	1	0	0	0	0	1	1	0	0	0
2	0	1	0	0	0	0	0	1	1	0	0	0
2	0	2	0	0	0	0	0	1	1	0	0	0
2	0	0	0	0	0	0	0	1	1	0	0	0
2	0	0	0	1	0	0	0	0	0	0	0	0
2	0	0	0	2	0	0	0	0	0	0	0	0
1	0	0	0	2	0	0	0	0	0	1	0	0
2	0	0	0	2	0	0	1	1	0	1	0	0
2	0	1	0	2	0	0	1	1	0	1	0	0
2	0	1	0	0	0	0	1	1	0	1	0	0
2	0	0	0	0	0	0	1	1	0	1	0	0
2	1	0	0	0	0	0	1	1	0	1	0	0
2	2	0	0	0	0	0	1	1	0	1	0	0
2	0	0	0	0	0	0	1	1	0	1	0	0
2	0	0	0	0	0	1	0	0	0	1	0	0
2	0	0	0	0	0	2	0	0	0	1	0	0
1	0	0	0	0	0	2	0	0	0	1	0	1
2	0	0	0	0	0	2	1	1	0	1	0	1
2	1	0	0	0	0	2	1	1	0	1	0	1
2	0	0	0	0	0	2	1	1	0	1	0	1
2	0	0	0	0	0	0	1	1	0	1	0	1
2	0	1	0	0	0	0	1	1	0	1	0	1
2	0	2	0	0	0	0	1	1	0	1	0	1
2	0	0	0	0	0	0	1	1	0	1	0	1
2	0	0	0	0	0	1	0	0	0	1	0	1
2	0	0	0	0	0	2	0	0	0	1	0	1
1	0	0	0	0	0	2	0	0	0	1	0	2
2	0	0	0	0	0	2	0	1	1	1	0	2
2	0	1	0	0	0	2	0	1	1	1	0	2
2	0	1	1	0	0	2	0	1	1	1	0	2
2	0	1	1	0	0	0	0	1	1	1	0	2
2	0	0	1	0	0	0	0	1	1	1	0	2
2	0	0	2	0	0	0	0	1	1	1	0	2
2	0	0	0	0	0	0	0	1	1	1	0	2
2	0	0	0	1	0	0	0	0	0	1	0	2
2	0	0	0	2	0	0	0	0	0	1	0	2
1	0	0	0	2	0	0	0	0	0	2	0	2
2	0	0	0	2	0	0	1	1	0	2	0	2
2	0	1	0	2	0	0	1	1	0	2	0	2
2	0	1	0	0	0	0	1	1	0	2	0	2
2	0	0	0	0	0	0	1	1	0	2	0	2
2	1	0	0	0	0	0	1	1	0	2	0	2
2	2	0	0	0	0	0	1	1	0	2	0	2
2	0	0	0	0	0	0	1	1	0	2	0	2
2	0	0	0	0	0	1	0	0	0	2	0	2
2	0	0	0	0	0	2	0	0	0	2	0	2
3	0	0	0	0	0	2	0	0	0	2	0	3
3	3	0	0	0	0	2	0	0	0	2	0	3
3	3	3	0	0	0	2	0	0	0	2	0	3
3	3	3	3	0	0	2	0	0	0	2	0	3
3	3	3	3	0	0	0	0	0	0	2	0	3
3	3	3	3	3	0	0	0	0	0	2	0	3
3	3	3	3	3	3	0	0	0	0	2	0	3
3	3	3	3	3	3	3	0	0	0	2	0	3

Conclusão

Com a realização deste trabalho conseguimos perceber o funcionamento e a importância dos mecanismos associados à execução e sincronização de processos e threads.

Bibliografia

Slides da disciplina

Documentação fornecida pelo DoxyFile